PATENT ABSTRACTS OF JAPAN

(11)Publication number : 07-281912
(43)Date of publication of application : 27.10.1995

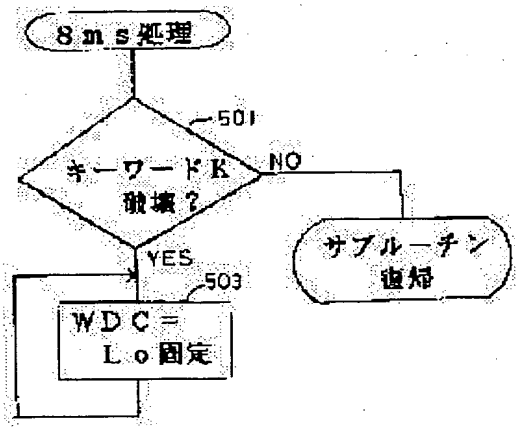
(51)Int.Cl. G06F 11/00
G06F 9/34
G06F 9/42

(21)Application number : 06-069628
(22)Date of filing : 07.04.1994

(71)Applicant : NIPPONDENSO CO LTD
(72)Inventor : KAMIYA TSUTOMU
KONDO HIROSHI

(54) STACK ABNORMALITY DETECTING DEVICE

(57)Abstract:
PURPOSE: To surely detect the overflow of a stack pointer without applying the large load to a CPU.
CONSTITUTION: It is decided at initialization whether or not a key word K writen in a key word area A0 of an address that is lower than a stack area by one decree is different from the contents of the present key word area (501). If not different, the processing returns directly to the original processing. If different, a signal WDC is fixed at a low level (503). A monitor part detects this state and forcibly resets a CUP. Thus the CPU is started again to carry out the processing in a normal state. Thus it is possible to prevent such a case where a stack pointer moves into an address lower than the area A0 to cause the runaway of a program or to destroy a RAM. Furthermore even an instantaneous stack abnormality can be detected since it remains as the destruction of the word K.



LEGAL STATUS

[Date of request for examination] 01.12.2000
[Date of sending the examiner's decision of rejection] 28.08.2001
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開平7－281912
(43)公開日　平成７年(1995)10月27日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　11/00	３１０　Ｈ			
9/34	３４０　Ｃ			
9/42	３３０　Ｃ			

審査請求　未請求　請求項の数５　ＯＬ　（全７頁）

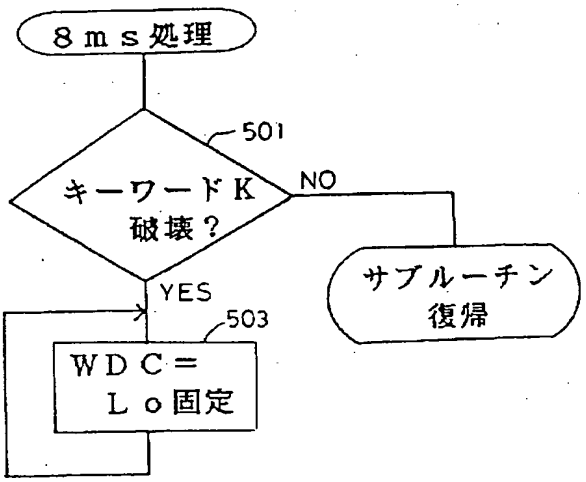
(21)出願番号　特願平6－69628

(22)出願日　平成６年(1994)４月７日

(71)出願人　000004260
日本電装株式会社
愛知県刈谷市昭和町１丁目１番地
(72)発明者　神谷　努
愛知県刈谷市昭和町１丁目１番地　日本電装株式会社内
(72)発明者　近藤　浩
愛知県刈谷市昭和町１丁目１番地　日本電装株式会社内
(74)代理人　弁理士　足立　勉

(54)【発明の名称】　スタック異常検出装置

(57)【要約】
【目的】　ＣＰＵに大きな負荷をかけることなく、スタックポインタがオーバフローを生じた場合を確実に捉える。
【構成】　イニシャル時にスタック領域より一つ低いアドレスのキーワード領域に書き込まれたキーワードＫと、現在のキーワード領域の内容と異なるかどうかを判別する（ステップ５０１）。異ならない場合はこのまま元の処理に復帰する。異なる場合は、ＷＤＣ信号を低レベルに固定する（ステップ５０３）。監視部３９がこの状態を検出してＣＰＵを強制リセットする。従って、ＣＰＵは再度立ち上げ直されて正常な状態で処理を開始することになる。このように、キーワード領域Ａ0より低いアドレスにスタックポインタが進んでプログラムが暴走したり、ＲＡＭが破壊されることを防止できる。しかも瞬間的なスタック異常が生じていてもキーワードＫの破壊として残存しているのでその検出が可能となった。



【特許請求の範囲】
【請求項１】　プロセッサが使用するスタック領域のオーバフローを検出するスタック異常検出装置であって、メモリ上のスタック領域に隣接するあるいは近傍のアドレスに設けられ、所定キーワードが設定されたキーワード領域と、
上記キーワード領域の設定内容が上記所定キーワードと同一か否かを監視し、同一でない場合はスタック異常と判定する判定手段と、
を備えたことを特徴とするスタック異常検出装置。
【請求項２】　上記キーワード領域が、スタック領域の最高点であるアドレスを越えたアドレスに設けられた請求項１記載のスタック異常検出装置。
【請求項３】　上記所定キーワードが、正常時にスタック領域に設定される可能性が低いまたは無い値である請求項１または２記載のスタック異常検出装置。
【請求項４】　上記所定キーワードが、スタック領域の最高点であるアドレスの次のアドレスに設けられている請求項２または３記載のスタック異常検出装置。
【請求項５】　上記キーワード領域が、複数のアドレスにわたって設けられている請求項１～４のいずれか記載のスタック異常検出装置。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明は、スタック異常検出装置に関し、特にコンピュータにおけるプログラムの実行に伴うスタックオーバフローの検出装置に関する。
【０００２】
【従来の技術】マイクロコンピュータ等において、外来ノイズによりＣＰＵの誤動作が発生した場合、スタックポインタの進みすぎ、いわゆるスタックオーバフローにより、他領域のメモリ内容を破壊する可能性がある。一般に、スタックオーバフローの異常を検出する方法として、スタック領域の上限アドレスと実際のアドレス（スタックポインタの値）とを比較し、実際のアドレスがこの上限アドレスを超えた時、スタックオーバフローと判断するものがある（例えば、特開平３－６６１８号）。
【０００３】
【発明が解決しようとする課題】しかしながら、上記技術は、ソフトウェアで実現する場合、ＣＰＵが一つのシステムでは、当然、常時連続モニタ不可能であるため、瞬間的にスタックポインタのアドレスが上限値を超えた時は異常検出できないという問題が有った。また、常時連続モニタを可能にするにはハードウェアによる手段が必要となるため、ソフトウェアで実現する場合と比べてコストが増加するという問題が有った。
【０００４】この他に、スタックポインタが変化する状況、即ち、ソフトウエア割込やハードウエア割込が生じた際に、スタックポインタの値が異常な値を示していないかを検出する技術が提案されている（特開平２－２９３９３９号）。しかし、この技術も割込処理が発生するたびに、スタックポインタチェック処理を実施するため、ＣＰＵに与える負荷が大きくなり、他の処理速度に影響するため採用し難いものであった。
【０００５】この様な問題点に鑑み、本発明は、ＣＰＵに大きな負荷をかけることなく、かつスタックポインタがオーバフローを生じた場合を確実に捉えることを目的としてなされたものである。
【０００６】
【課題を解決するための手段】請求項１記載の発明は、プロセッサが使用するスタック領域のオーバフローを検出するスタック異常検出装置であって、メモリ上のスタック領域に隣接するあるいは近傍のアドレスに設けられ、所定キーワードが設定されたキーワード領域と、上記キーワード領域の設定内容が上記所定キーワードと同一か否かを監視し、同一でない場合はスタック異常と判定する判定手段と、を備えたことを特徴とするスタック異常検出装置である。
【０００７】請求項２記載の発明は、上記キーワード領域が、スタック領域の最高点であるアドレスを越えたアドレスに設けられた請求項１記載のスタック異常検出装置である。請求項３記載の発明は、上記所定キーワードが、正常時にスタック領域に設定される可能性が低いまたは無い値である請求項１または２記載のスタック異常検出装置である。
【０００８】請求項４記載の発明は、上記所定キーワードが、スタック領域の最高点であるアドレスの次のアドレスに設けられている請求項２または３記載のスタック異常検出装置である。請求項５記載の発明は、上記キーワード領域が、複数のアドレスにわたって設けられている請求項１～４のいずれか記載のスタック異常検出装置である。
【０００９】
【作用及び発明の効果】請求項１記載の発明は、メモリ上のスタック領域に隣接するあるいは近傍のアドレスにキーワード領域が設けられている。このキーワード領域には、所定キーワードが設定されている。そして判定手段が、上記キーワード領域の設定内容が上記所定キーワードと同一か否かを監視している。同一でない場合は判定手段はスタック異常と判定する。
【００１０】このことにより、万一、何等かの原因でスタックがオーバフロー等により、上記キーワード領域をスタックポインタが指した場合、当然そのアドレスに何等かのデータが書き込まれることになる。この後、例えスタックポインタが正常なスタック領域を指す状態になったとしても、キーワード領域が書き換えられているという状態は残っている。したがって、後で、そのキーワード領域にアクセスして内容を調べることにより、スタックポインタの値の異常が有ったことが判明する。即ちスタックポインタが変化するその瞬間、あるいはその直

後に調査しなくても、オーバフロー等の異常の履歴が残存しているので、ＣＰＵに過大な負担をかけない範囲でスタックポインタの過去の異常のチェックが可能となる。このように、常時スタックポインタを監視しているのと同等の効果を、ＣＰＵに過大な負荷をかけることなく、実施することができる。

【００１１】上記所定キーワードは、スタックポインタの異常時に書き換えられるコードと偶然一致する可能性はかなり低い。このため、通常用いられるコード以外で有れば、即ち、例えばプログラムに用いられていない値（コード）、正常時にスタック領域に設定される可能性が低い値（コード）あるいは正常時にスタック領域に設定されることが無い値（コード）で有れば、いかなるコードを所定キーワードとして設定しても本発明の目的を達成できる。特に、予め、プログラムのデータも含めた全コードをチェックしておき、そのコード以外のコードを所定キーワードとして設定してもよい。また、プログラム自体を、プログラムの起動時にプログラム内のデータも含めた全コードをチェックして、存在しないコードを検出する処理と、この処理により得られた存在しないコードをキーワード領域に設定する処理とを設けたプログラムとして、プログラム自身で、使用されるコード以外のコードを所定キーワードとして、自動設定するようにしてもよい。

【００１２】キーワード領域が設定されるアドレスは、上記スタック領域に隣接するあるいは近傍のアドレスであればよい。例えば、スタック領域の最高点であるアドレスを越えたアドレスでもよい。この場合、スタック領域の最高点であるアドレスの次のアドレスでもよく、それよりも更に先のアドレスでもよい。スタックは、アドレスの高い方から低い方に積まれて行くので、上記スタック領域の最高点を越えたアドレスとは最高点より低いアドレスである。「スタック領域の最高点であるアドレスの次のアドレス」とは、最高点より一つ低いアドレスであり、「それよりも更に先のアドレス」とは、もっと低いアドレスを指している。

【００１３】また逆にスタックの起点となるスタートアドレスよりも高いアドレスにキーワード領域を設けてもよい。例えば、スタートアドレス＋１のアドレスをキーワード領域としてもよく、それよりも更に高いアドレスをキーワード領域としてもよい。これは逆にアドレスが戻りすぎる異常の検出に有効である。

【００１４】また、このキーワード領域も、一つのアドレス領域のみでなく、複数のアドレスにわたってもよく、また連続して存在せずに不連続にキーワード領域を設定してもよい。このようにすることにより、スタックポインタの各種の異常の検出に有効であり、また書き換えられている領域の広さからその異常の程度も判定できる。

【００１５】【実施例】以下、この発明をエンジン制御装置に具体化した実施例について図面に基づいて説明する。

［実施例１］図１は、実施例のエンジン制御装置を示すブロック図である。同図に示すように、制御回路１内に設けられたマイクロコンピュータ３は、ＣＰＵ（中央処理装置）５，リードオンリメモリ（ＲＯＭ）７，ランダムアクセスメモリ（ＲＡＭ）９，タイマー１１および入出力ポート１５を備えている。また、これらの構成要素はバス１３を介して接続されており、このバス１３を介して入出力データの転送が行われるようになっている。ＲＯＭ７内には、エンジン制御用プログラム、その他の処理プログラムおよびそれらの演算処理に必要な種々のデータが予め格納されている。タイマー１１はフリーランニングカウンタ，アウトプットコンペアレジスタ等を有する。なお、図１には示していないが、マイクロコンピュータ３は、周知の入出力制御回路，メモリ制御回路等を備えている。

【００１６】また、制御回路１において、エアフローメータ，負圧センサ，水温センサ等のセンサ群１７からの電圧信号は、アナログマルチプレクサを含むＡ／Ｄ変換器１９に送り込まれる。そして、その後前記電圧信号がＡ／Ｄ変換器１９にて所定の変換周期で順次２進の信号に変換され、入出力ポート１５に送り込まれる。また、スタータスイッチ，スロットルセンサ等ＳＷ群２１からの信号は入力バッファ２３に入力された後、入出力ポート１５に送り込まれる。基準位置センサ，クランク角センサ等タイミングセンサ群２５からの角度位置信号は、波形整形回路２７で矩形状の基準位置信号及びクランク角信号に波形整形された後、入出力ポート１５に送り込まれる。

【００１７】このようにして取り込まれたデータに基づいて、プログラムによりＣＰＵ５にて、所定の演算処理が実行される。この演算結果に基づいて生成された出力データが、入出力ポート１５，出力回路３１を介してアクチュエータ群２９に送り込まれる。このことにより所望のエンジン制御が実現されている。

【００１８】ＣＰＵ５は図２に示すごとく、スタックポインタ３３を含むレジスタ群３５を有する。スタックポインタ３３は図３に示すごとく、ＲＡＭ９内に割り当てられたスタック領域３７内のいずれかのアドレスを指定する。ＣＰＵ５は必要に応じてスタックポインタ３３で示されている位置のスタック領域３７からデータを出し入れする。

【００１９】制御回路１は上述した構成以外に、図４に示すＣＰＵ５の動作状態をモニタする監視部３９を有する。この監視部３９はＣＰＵ５が何らかの理由で動作停止する、もしくは無限ループに入るとＷＤＣパルスを出力できなくなるので、それを検出した監視部３９がＣＰＵ５を強制リセットする。このような監視部３９はウォッチドッグタイマとして一般的に知られている。

【００２０】次に、ＣＰＵ５により実行される制御を、図５のフローチャートに基づいて説明する。イグニッションスイッチ（ＩＧＰ）オンによるリセット後に開始されるメインループのイニシャル処理内で、スタックポインタ３３初期化により、スタックポインタ３３にはスタートアドレス（図３のＡs0の位置）が設定される（ステップ１０１）。その後、図３に示すごとく、本プログラム内でスタックポインタ３３が進む最高点（スタック領域中最低アドレス）Ａsnの次のアドレスＡ0、すなわちスタックポインタ３３が到達し得ないアドレスにキーワードＫを書き込む（ステップ１０３）。

【００２１】ここでキーワードＫは、プログラム内容を調べてスタック領域３７に書かれない値を設定しておく。具体的には割込処理、サブルーチン・コール時の戻り先アドレス、及び回避データ、またプログラム実行時にワークＲＡＭとしてスタック領域に格納したデータが取り得ない値である。この様な値が存在しない場合は、出現頻度が最低のデータ値を設定する。このようにプログラム内容を調べる処理は、予めプログラム設計時に完了してプログラム自体にそのデータを組み込んでおいてもよい。またプログラムの内容を調べて、スタック領域に書かれることがない値を設定する、あるいは頻度が最低の値を設定する処理をプログラム自体に組み込んでおき、キーワードＫの書き込み処理（ステップ１０３）の前に実行して、適切なキーワードＫを決定できるようにしてもよい。

【００２２】上述したイニシャル処理（ステップ１０１，１０３等）の終了後、メイン処理１０５のループに入る。メイン処理１０５は複数のサブルーチン（Ｓｂ１，Ｓｂ２，…）を含んで構成される。また、メイン・ループ処理中にタイマー１１等が割り込み処理（Ｉｎｔ１，…）を起動する。またサブルーチン処理や割り込み処理中には図５に示すごとくネスティングが存在する。

【００２３】図７を参照して、スタックポインタ３３の動作例を説明する。イニシャル終了後、またはメイン処理が次の周回に入った時、正常で有れば、当然、スタックポインタ３３はスタック領域のスタート・アドレスＡs0を指す。そしてメイン処理１０５の中でサブルーチンＳｂ１がコールされた時、コール元アドレスをスタート・アドレスＡs0に格納してスタックポインタ３３は次のアドレスＡs1を指す（ステップ２０１）。尚、本実施例では１６ビットＣＰＵの処理なのでスタックポインタはアドレスが２づつ変化される。

【００２４】その後、割り込みが発生して割り込み処理Ｉｎｔ１により、割り込まれた元アドレスをアドレスＡs1に格納してスタックポインタ３３は次のアドレスＡs2を指す（ステップ３０１）。更に、割り込み処理Ｉｎｔ１内で各種レジスタの内容をＰＵＳＨ命令（退避させるレジスタ等の内容をスタックに積む命令）を実行する（ステップ３０３）ため、スタックポインタ３３はＡs2からＡs3に移動する。その後、割り込み処理Ｉｎｔ１内からサブルーチンＳｂ２がコールされて（ステップ３０５）、スタックポインタ３３はＡs4を指す（ステップ４０１）。その後、サブルーチンＳｂ２の終了時にスタックポインタ３３はＡs4からＡs3に戻り（ステップ４０３）、更に、割り込み処理Ｉｎｔ１で、ＰＵＬＬ命令（退避されていたレジスタ等の内容を復帰させる命令）が実行されて、スタックポインタ３３はＡs3からＡs2に戻り（ステップ３０７）、更に割り込み処理Ｉｎｔ１終了時にスタックポインタ３３はＡs2からＡs1に戻る（ステップ３０９）。更に、サブルーチンＳｂ１に戻った後、その終了時にスタックポインタ３３はＡs1からＡs0に戻り（ステップ２０３）、メイン処理に帰って来る。

【００２５】このような処理を繰り返している間に、スタックポインタ３３の値がいくつか余分に進んでしまう異常が発生することがある。例えば、プログラム動作中にノイズ等何らかの理由によりサブルーチン、割り込み処理がコールされる前の状態に復帰しない様な誤動作が発生したり、上述したＰＵＳＨ命令後ＰＵＬＬ命令が実施されなかった様な場合である。このような事態が生じると、その時点あるいはそれ以後のスタックポインタ３３の変更処理でスタック領域３７の最高点であるアドレスＡsnを越える場合がある。このような場合には、アドレスＡsnの一つ下のキーワード領域Ａ0をスタックポインタ３３が指すことになり、この状態で割込処理、サブルーチンコールあるいはデータの退避処理が実行されると、キーワード領域Ａ0に帰り先のアドレスや退避データが書き込まれてしまい、キーワードＫは破壊されてしまう。

【００２６】この状態は図６に示すスタック異常検出処理にて、破壊されたか否かが判定されて、破壊されている場合は必要な処理が実行される。スタック異常検出処理はメイン処理の１つであり８ｍｓ毎にコールされるサブルーチンとし、イニシャル時にステップ１０３の処理で書き込まれたキーワードＫと現在のキーワード領域Ａ0の内容と異なるかどうかを判別する（ステップ５０１）。異ならない場合はこのまま元の処理に復帰する。異なる場合はＷＤＣ信号のＬｏセットを無限に繰り返す処理（ステップ５０３）に移る。即ち、図４に示したＷＤＣ信号が低レベルに固定されてしまうことになる。このようにＣＰＵ５は正常にＷＤＣパルスを出力できないため、監視部３９がこの状態を検出してＣＰＵ５を強制リセットする。従って、ＣＰＵ５は再度立ち上げ直されて正常な状態で処理を開始することになる。

【００２７】このように図６の処理により、キーワード領域Ａ0より低いアドレスにスタックポインタ３３が進んで、サブルーチンがコール元に戻らずに暴走したり、ＲＡＭ領域が破壊されることを防止できる。しかもハードウェア手段を必要とせず瞬間的なスタック異常が生じていても、キーワード領域Ａ0にキーワードＫの破壊と

して残存しているのでその検出が可能となった。

［他の実施例］実施例２として、キーワード領域設定の他の例を示す。図３にＡｘで示すように、キーワード領域のアドレス（１６ビット分）も一つでなく、２つでもそれ以上でも良い。このようにキーワード領域を複数のアドレスに設定し、その全領域をステップ５０１でチェックして、そのアドレスの一つでも破壊されているキーワードが発見されれば、ステップ５０３を実行すれば良い。こうすれば、万一、領域Ａ0を飛び越すようなスタックポインタ３３の異常が生じても発見することができ、異常発見の確率が増大する。例えば、いかなる値をもスタック領域の値が取り得る場合等に発見精度を上げることができる。

【００２８】実施例３として、スタックポインタ３３が進む最高点（スタック領域中の最低アドレス）の次のアドレスにキーワードＫを置いているが、図３にＡｙで示すように、スタックポインタ３３のスタートアドレス＋１のアドレスにもキーワードＫを置いても良い。更にこのようにスタック領域３７よりも高い方のアドレスにキーワード領域を配置する場合も、複数アドレスにわたってキーワード領域を配置しても良く、それらをステップ５０１でチェックして、そのアドレスの一つでも破壊されているキーワードＫが発見されれば、ステップ５０３を実行すれば良い。こうすれば、逆にアドレスが戻りすぎる異常の検出に有効である。

【００２９】実施例４として、実施例１，２，３を組み合わせたキーワード領域の配置としても良い。このようにすれば、より一層異常発見の制度が向上する。尚、各実施例において、図６のチェックプログラムを８ｍｓ毎の割込みとしているがこれに限定されない。チェックプログラムはメインルーチンに直接配置してもよく、更に、ＣＰＵ５の負担が過大にならない限りサブルーチンや割込ルーチンに配置しても良い。また、ＣＰＵ５が割り込んでもエンジン制御に影響のないタイミングを選択し、割り込みにて図６の処理を実行してもよい。

【００３０】また、キーワードＫとしては、異常時にキーワードＫと同じ値（コード）がキーワード領域に書き込まれるのを防止するため、正常時にスタック領域３７に設定される可能性が低い値あるいは正常時にスタック＊領域３７に設定されることが無い値が設定される。このようなキーワードＫを決定する場合、１６ビット（２バイト）のキーワードＫそのものの設定が行われない可能性をチェックしても良いが、８ビット（１バイト）としてスタック領域３７に設定される可能性の低いあるいは無いコードを決定し、それを２つ並べて、１６ビットのキーワードＫとしても良い。また、その８ビットを反転して、反転前の８ビットと反転後の８ビットとを並べて、１６ビットのキーワードＫとしても良い。また、８ビット自体がスタック領域３７に設定される可能性が低いまたは無いならば、これに他のいかなる８ビットを組み合わせて構成した１６ビットでもよい。またその８ビットは、スタック領域３７に設定される可能性が低いまたは無いのであるから、その８ビットに対して、その８ビットに所定の演算をして求めた８ビット、あるいは予め決定しておいた他の適当な８ビットコードを組み合わせて１６ビットのキーワードＫとしても良い。

【００３１】上記各実施例において、ステップ５０１の処理が判定手段としての処理に該当する。ＣＰＵ５として、１６ビットタイプを用いたが、８ビットタイプ、３２ビットタイプ等のいずれを用いてもよい。

【図面の簡単な説明】

【図１】　実施例のエンジン制御装置を示すブロック図である。

【図２】　レジスタ群の構成説明図である。

【図３】　ＲＡＭ内のスタック領域周辺の説明図である。

【図４】　ＣＰＵとＣＰＵの動作状態をモニタする監視部との関係を示すブロック図である。

【図５】　ＣＰＵが実施する処理の一例を示すフローチャートである。

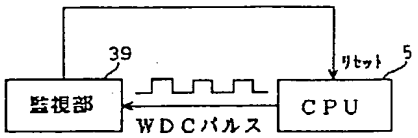
【図６】　スタック異常検出処理のフローチャートである。

【図７】　スタックポインタの動作例の説明図である。

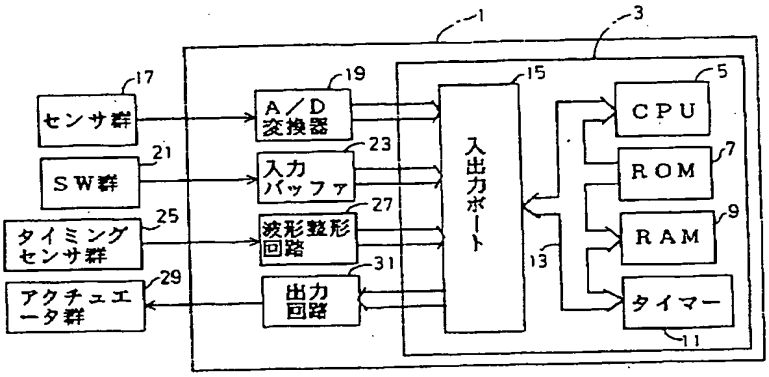
【符号の説明】

Ａ0…キーワード領域	１…制御回路
３…マイクロコンピュータ	５…ＣＰＵ
３３…スタックポインタ	３５…レジスタ群
３７…スタック領域	３９…監視部

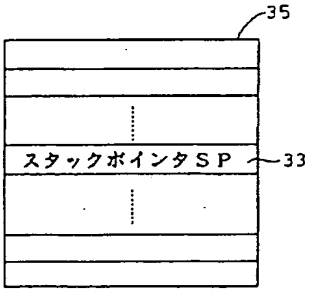
【図４】



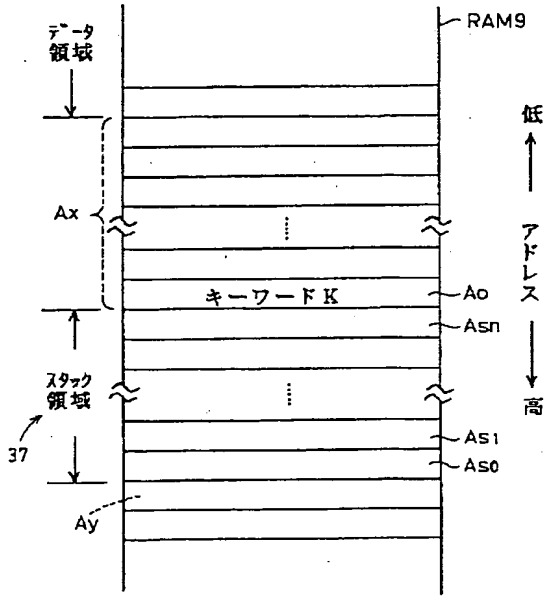
【図１】


1
3
17
センサ群
19
A／D
変換器
15
5
CPU
21
SW群
23
入力
バッファ
7
ROM
入出力ポート
25
タイミング
センサ群
27
波形整形
回路
9
RAM
13
29
アクチュエータ群
31
出力
回路
タイマー
11

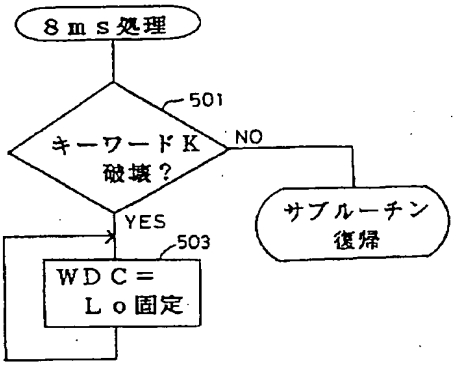
【図２】


35
スタックポインタSP
33

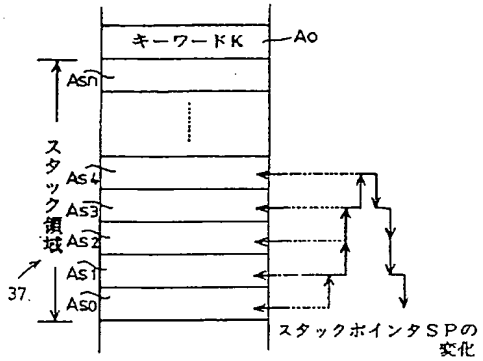
【図３】


データ
領域
RAM9
低
アドレス
高
Ax
キーワードK
Ao
Asn
スタック
領域
37
As1
As0
Ay

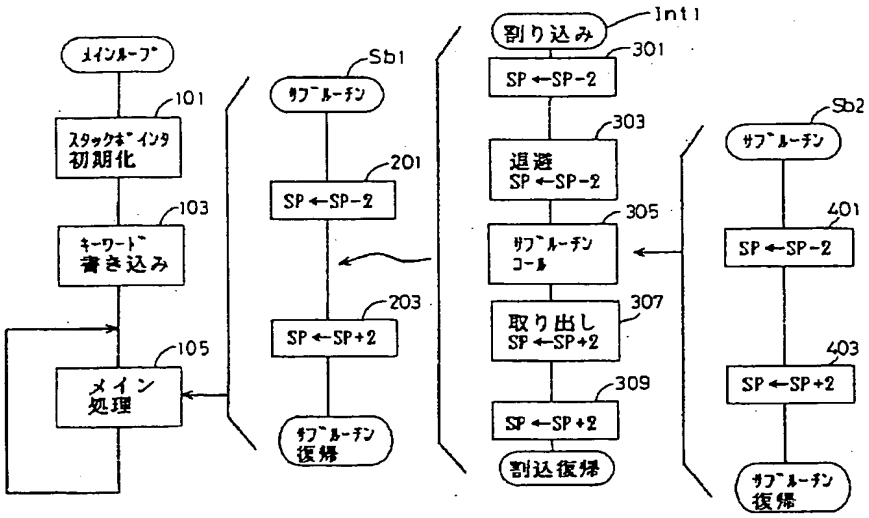
【図６】


8ms処理
501
キーワードK
破壊？
NO
サブルーチン
復帰
YES
503
WDC＝
Lo固定

【図７】


キーワードK
Ao
Asn
スタック領域
As4
As3
As2
As1
As0
37
スタックポインタSPの
変化

【図５】


メインルーフ
101
スタックポインタ
初期化
103
キーワード
書き込み
105
メイン
処理
Sb1
サブルーチン
201
SP←SP-2
203
SP←SP+2
サブルーチン
復帰
Int1
割り込み
301
SP←SP-2
303
退避
SP←SP-2
305
サブルーチン
コール
307
取り出し
SP←SP+2
309
SP←SP+2
割込復帰
Sb2
サブルーチン
401
SP←SP-2
403
SP←SP+2
サブルーチン
復帰